GeekBrains

Введение во Vue.js

Курс Vue.js

Оглавление

Введение

Первый вопрос который задают себе начинающие разработчики, это для чего мне изучать фреймворки, если я уже знаю javascript и с помощью данного языка программирования можно реализовать весь функционал на странице, поэтому первое с чего хочется начать, это понимание – а нужен ли нам фреймворк?

Сложные пользовательские интерфейсы включают в себя работу со списками, формами, управляющими элементами (такими как - кнопки слайдера и так далее). Как правило интерфейсы строятся из отдельных блоков, каждый из которых имеет какой-то интерактивный функционал, но сложность возникает при композиции нескольких элементов на странице или даже организации многостраничного приложения.

Пользовательские интерфейсы не сразу стали сложными. 10-15 лет назад достаточно было иметь статический сайт с формой для отправки заявки или заказа. С тех пор индустрия сильно развилась, а вместе с ней также выросли ожидания пользователей по возможностям, которые должна предоставлять веб-страница.

Часто бывает, что задачи, которые должно решать веб-приложение, основываются на сложной логике. Учитывая это, реализация такого сложного функционала требует применения нескольких парадигм для управления данными.

Какие именно проблемы возникают:

- отображение на странице переменных JS, которые могут меняться по какой-то логике
- действия, которые выполняют пользователи - клики по кнопкам, ввод данных в форме, - все это должно обрабатываться логикой скриптов
- необходимость упростить создание сложных приложений путем их разбиения на логические и функциональные блоки
- обеспечение воспроизводимости поведения - при выполнении одних и тех же действий мы должны получать тот же самый результат - и визуально на странице, и в модели данных
- отработка асинхронных процессов, которые могут выполняться параллельно, например загрузка данных и обработка других действий пользователя.

Учитывая озвученные проблемы, разрабатывать сложное приложение, которое будет обладать большим функционалом, может быть достаточно сложно. Чтобы упростить себе жизнь, можем воспользоваться специальной платформой, которая позволит облегчить разработку отдельных программных компонентов и объединить их в один проект.

По своей сути, мы сейчас почти что дали определение понятию “фреймворк”. В JavaScript существует множество фреймворков различной степени функциональности и, соответственно, популярности. В рамках нашего курса мы с вами рассмотрим фреймворк Vue.js

Для ускорения разработки приложений используются фреймворки. Это готовые наборы решений самых популярных задач. Благодаря фреймворкам разработчику не нужно думать о том, как организовать структуру приложения. У него уже есть все основные инструменты для рутинных задач: обработки событий, разделения приложения на компоненты и связывания компонентов между собой.

На конец 2022 года самые популярные JavaScript-фреймворки – **Angular**, **React** и **Vue** от независимого разработчика Эвана Ю. У каждого из этих фреймворков есть плюсы и минусы, но в целом все они подходят для создания приложений. Мы подробно остановимся на Vue.js.

Vue, как написано на главной странице проекта - это "прогрессивный JavaScript фреймворк". Vue создан, чтобы дать возможность быстро создавать сложные пользовательские интерфейсы.

Сотрудник Google Эван Ю выпустил первую версию Vue.js в октябре 2015 года. По его задумке, фреймворк должен был стать инструментом для быстрого прототипирования сложных интерфейсов в противовес сложному Angular и сырому в то время React. Фреймворк быстро набрал популярность. Сейчас он поддерживается компаниями и сообществами.

Прогрессивным он называется, потому что использовать все его возможности совершенно необязательно. Программист может использовать функционал фреймворка по мере усложнения задач, которые решает разрабатываемое приложение.

Например, на начальном этапе разработчик может даже не использовать webpack для сборки кода, а просто подключать фреймворк через самый обычный тег script. И только по мере развития проекта перейти на какой-нибудь сборщик. И так с любым инструментарием, предоставляемым фреймворком. Его возможности прогрессивно раскрываются с ростом потребностей разработчиков - мы увидим это не раз на протяжении курса.

Создатель фреймворка начал работу над проектом в 2015 году. Эван Ю, до этого работал в команде Google которая развивала другой популярный тогда проект AngularJS, поэтому некоторые из решений во Vue напомнят вам Angular если вы с ним работали.

Особенно популярен стал Vue среди разработчиков в 2017 когда вышла вторая мажорная версия фреймворка.

Установка

Vue.js работает во всех браузерах, начиная с Internet Explorer 8. На официальном сайте проекта (https://ru.vuejs.org/v2/guide/) можно скачать одну из двух версий: для разработки и для использования в готовом приложении. Обе работают одинаково, разница в том, что в версии для разработки удобный для чтения исходный код и служебные извещения, которые помогают отлаживать приложение.

Чтобы не скачивать исходники, подключим Vue.js через CDN: сам файл будет находиться на удалённом сервере, а мы подключим его к проекту как обычный внешний скрипт. Создадим файл index.html и добавим в него тег <script> со ссылкой на файл:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Vue.js</title>
</head>
<body>
  <script
src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
</body>
</html>
```

Теперь добавим отдельно файл script.js, в котором будем писать наше приложение. Подключим этот файл в index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Vue.js</title>
</head>
<body>
  <script
src="https://cdn.jsdelivr.net/npm/vue@2/dist/vue.js"></script>
  <script src="script.js"></script>
</body>
</html>
```

Теперь проверим, что **Vue.js** успешно подключен. Попробуем создать экземпляр класса **Vue**:

```
//script.js
const app = new Vue();
console.log(app);
```

Если всё работает верно, в консоли появился объект класса **Vue**:

```
Vue {_uid: 0, _isVue: true, __v_skip: true, _scope: EffectScope,
$options: {…}, …}
```

Теперь библиотека работает, и можно начинать создавать приложение.

Шаблоны

Прежде всего нужно научиться подставлять данные в html-шаблон. Чтобы указать, какая именно часть нашего html должна восприниматься Vue как шаблон, добавим блок с id="app" и привяжем его к объекту app. При создании экземпляра класса Vue ему передаётся объект с настройками. Поле el этого объекта как раз отвечает за привязку к html-элементу:

Первым делом мы определяем с каким блоком мы будем работать в html

```
...
<body>
  <div id="app"></div>
  ...
</body>
...
```

```
//script.js
const app = new Vue({
  el: '#app'
});
```

Теперь можно добавить данные в объект **app**. Данные передаются в поле **data** объекта настроек:

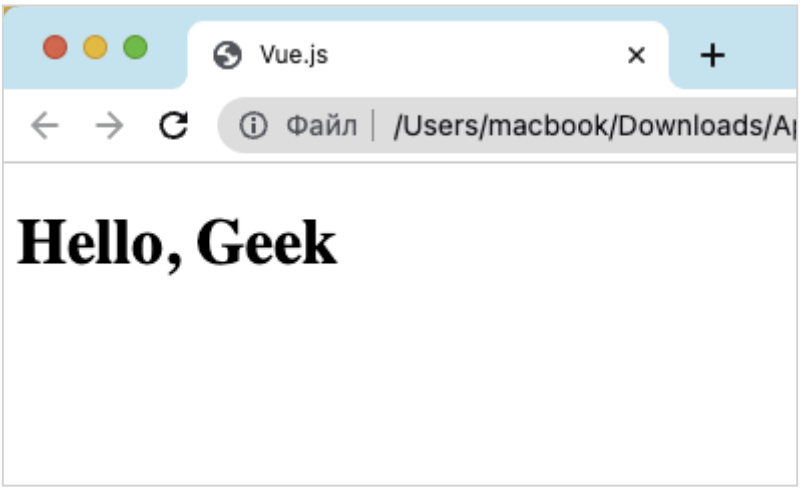
```
//script.js
const app = new Vue({
```

```
  el: '#app',
  data: {
    name: 'Geek'
  }
});
```

К данным можно обратиться из html, обернув имя поля в двойные фигурные скобки. Такая запись называется mustache-синтаксисом:

```
...
<div id="app">
  <h1>Hello, {{ name }}</h1>
</div>
...
```

Теперь если мы откроем страницу в браузере, то увидим надпись «Hello, Geek».



Запись внутри фигурных скобок – обычный JavaScript, а name – переменная. Поэтому любые методы и операции работают и внутри шаблона:

```
...
<div id="app">
  <h1>Hello, {{ name.toUpperCase() }}</h1>
</div>
...
```

В этом случае на странице мы увидим «Hello, GEEK».

Вот мы и создали наше первое Vue-приложение! Выглядит как простая отрисовка шаблона, но «под капотом» Vue выполнил немало работы. Данные и DOM теперь реактивно связаны. Как это проверить? Просто откройте консоль JavaScript в браузере (на созданной странице) и задайте свойству app.name новое значение. Вы тут же увидите соответствующее изменение в браузере.

Обратите внимание, что теперь больше не нужно напрямую взаимодействовать с HTML. Приложение Vue присоединяется к одному элементу DOM (#app в данном случае), а затем полностью контролирует его. HTML является нашей точкой входа, но всё остальное происходит внутри вновь созданного экземпляра Vue.

События

Для обработки отслеживания событий на элементе используется атрибут **v-on**. Но прежде чем навешивать (то есть связывать с элементом) событие, нужно позаботиться об обработчике. Добавим поле methods в объект настроек. Там будут перечислены функции, доступные из шаблона:

```
//script.js
const app = new Vue({
  el: '#app',
  data: {
    names: ['Frodo', 'Sam', 'Meriadoc', 'Peregrin']
  },
  methods: {}
});
```

Добавим метод обработки клика **clickHandler()**. Пока что будем просто выводить в консоль слово **click**:

```
const app = new Vue({
  el: '#app',
  data: {
    names: ['Frodo', 'Sam', 'Meriadoc', 'Peregrin']
  },
  methods: {
    clickHandler() {
      console.log('click');
    }
  }
});
```

Теперь привяжем вызов этого метода к клику на элемент списка:

```
...
<ul>
  <li v-for="name in names" v-on:click="clickHandler">{{ name
}}</li>
</ul>
...
```

Запись можно немного сократить, заменив **v-on** на **@**:

```
...
<ul>
  <li v-for="name in names" @click="clickHandler">{{ name }}</li>
</ul>
...
```

Разумеется, отслеживать можно не только клик, но и другие события, например @keyup. о них вы узнаете немного позже.

Иногда нужно выполнить какое-то действие сразу, как только загрузится приложение, не дожидаясь действий пользователя. Для этого можно использовать метод mounted. Эта функция записывается в одноименное поле объекта настроек и срабатывает сразу после загрузки приложения.

```
const app = new Vue({
  el: '#app',
  data: {
    names: ['Frodo', 'Sam', 'Meriadoc', 'Peregrin']
  },
  methods: {
    clickHandler() {
      console.log('click');
    }
  },
  mounted() {
    // Сработает сразу
  }
});
```

Условия

Наиболее частой проблемой при создании контента является условный рендеринг элементов. В зависимости от значения тех или иных данных мы хотим либо отобразить какой-то элемент, либо скрыть его.

Давайте рассмотрим пример Vue позволяет скрывать и показывать элементы в зависимости от условий. Если в примере

```
<div id="app">
   <input type="text" v-model="name" />
   <p>Hi, my name is {{ name }}</p>
</div>
```

удалить значение поля name, на странице всё равно останется надпись «Hi, my name is». Будем скрывать её, если в name записана пустая строка. Для этого к тегу <p> добавим атрибут **v-if**:

```
<div id="app">
   <input type="text" v-model="name" />
   <p v-if="name.length != 0">Hi, my name is {{ name }}</p>
</div>
```

Теперь элемент **<p>** будет отображаться только тогда, когда длина строки, записанной в **name**, не равна 0. Причем элемент не просто скрывается, когда условие **v-if** не выполняется, он полностью отсутствует в DOM.

Директива v-if, которая в зависимости от истинности выражения, переданного в нее, либо вставляет свой элемент в DOM либо удаляет его. Если error является пустой строкой (что в JavaScript эквивалентно false), то сообщение не будет отрендеренное, иначе блок будет вставлен в DOM дерево. Важно отметить, что данная директива именно удаляет и вставляет элемент.

Мы можем воспользоваться также директивой **v-show**. Она работает очень похоже, только вместо вставки и удаления элемента, она скрывает его, оставляя в DOM дереве. Изменение видимости элемента происходит путем переключения свойства display в none и обратно.

Подведем итоги

Как мы можем заметить, ни установка, ни изучение первых шагов не вызывает вопросов и проблем, в этом и есть основное преимущество vue.js он сочетает в себе простоту и функциональность. Самое главное правило, это конечно же побольше практиковаться и применять новые возможности языка, но о них вы узнаете в последующих уроках.